Capítulo V. Implementación del sistema.

En el capítulo anterior se mostró de forma gráfica por medio de la herramienta UML tanto la estructura de cada clase, como la forma de comunicación entre ellas. En este capítulo se profundizará sobre los módulos más importantes de cada una de las clases y se explicará de forma breve tanto su funcionamiento como la importancia que tienen dentro del sistema de detección de movimiento.

Dada la arquitectura del sistema, éste es dividido en tres grupos, sin embargo, en éste capítulo se tocarán solo dos de los tres grupos. Estos son el grupo de control y el grupo de modelo, debido a que son éstos mismos quienes realizan las tareas más importantes. Finalmente se mostrará el resultado que ofrece el grupo de la vista y sus interfaces.

5.1 Servlets del Control.

El control está compuesto por cuatro servlets que se encargan de recibir las peticiones del usuario y enviarlas al modelo para resolver dichas peticiones. A continuación se resaltan los módulos de código más importantes de cada uno de los servlets.

5.1.1 CargaImagen.

Este servlet recibe por parte del cliente un archivo, en este caso una imagen, así como el nombre con el cual será almacenado en el servidor. Su función es parsear el contenido de la petición y determinar las partes principales de la misma.

```
try {
        MultipartParser mp = new MultipartParser(
                        (javax.servlet.http.HttpServletRequest)request,10*1024*1024);
        Part part;
        while ((part = mp.readNextPart()) != null) {
                String name = part.getName();
                if (part.isParam()) {
                        contador++;
                        ParamPart paramPart = (ParamPart) part;
                        nombreArchivo = paramPart.getStringValue();
                        if(nombreArchivo == null)
                                nombreArchivo = "temporal.jpg";
                dir="C:/Tomcat/webapps/Tesis/Imagenes/"+nombreArchivo;
                }
                else if (part.isFile()){
                        FilePart filePart = (FilePart) part;
                        String fileName = filePart.getFileName();
                        if (fileName != null) {
                                try{
                                        byte gif[]=null;
                                        gif=fetchBytes(filePart.getInputStream());
                                        try{
                                                direcciones[cont] = dir;
                                                cont++;
                                                FileOutputStream fow =
                                                                new FileOutputStream(dir,
                                                false);
                                                fow.write(gif, 0, gif.length);
                                                fow.close();
                                                guarda(nombreArchivo, dir);
                                        }catch(IOException ioe){
```

```
                                        System.out.println("Error "+ioe);
                                }
                        }catch (Exception e){
                                e.printStackTrace();
                        }
                }
                else {
                        System.err.println("file; name=" + name +"EMPTY");
                }
        }
    }
}catch (IOException lEx) {
    lEx.printStackTrace();
}
```

Posteriormente, guarda la imagen en la base de datos del servidor para su procesamiento posterior.

```
try {
    Class.forName("org.gjt.mm.mysql.Driver");
    String url="jdbc:mysql://localhost:3306/tesis";
    Connection con = DriverManager.getConnection(url);
    Statement stmt = con.createStatement();
    stmt.executeUpdate("insert into imagenes values('"+nombre+"', '"+dir+"');");
    stmt.close( );
    con.close( );
}
catch( Exception e) {
    e.printStackTrace( );
}
```

Una vez realizado lo anteriormente mencionado, da paso al servlet *Comprobación* el cual se encarga de la comunicación con el grupo del control.

5.1.2 ImagenesDisponibles.

Presenta al usuario una barra de opción con todas las imágenes cargadas en la base de datos del servidor para su procesamiento. Este servlet realiza esta operación llenando un vector por medio de la siguiente consulta a la base de datos:

```
try {
        Class.forName("org.gjt.mm.mysql.Driver");
        String url="jdbc:mysql://localhost:3306/tesis";
        Connection con = DriverManager.getConnection(url);
        Statement stmt = con.createStatement();
        ResultSet rs;
        rs = stmt.executeQuery ("select distinct nombre from imagenes;");
        while (rs.next( ))  {
                String op = rs.getString(1);
                v.add(op);
        }stmt.close( );
        con.close( );
} catch( Exception e) {
        e.printStackTrace( );
}
```

Una vez que el usuario seleccionó la o las imágenes a procesar, da paso al servelt *Comprobacion.*

5.1.3 Comprobacion.

Este servlet recibe como parámetros de entrada la imagen o las imágenes a procesar, así como el método de procesamiento. Para la comunicación con el modelo utiliza el formato de sistemas expertos para el cual el pseudo-código es:

```
if(op.equals("metodo de procesamiento")){
        try{
                Image i = Toolkit.getDefaultToolkit().getImage(dir[0]);
                res = método(i);
                guardar(res);
        }
        catch(Exception e){
                System.out.println("Error invirtiendo imagen");
        }
}
```

Una vez resuelto el problema por el modelo, es este mismo servlet quien genera la página Web dinámica para mostrar al cliente el resultado del procesamiento.

5.2 Operadores del modelo.

El procesamiento básico de imágenes puede hacerse de dos formas, por medio de la alteración del valor de un píxel en una posición dada (individual) o bien basándose en operaciones de múltiples puntos [Pajares, et al., 2004].

La generación de un nuevo píxel para una nueva imagen se obtendrá por medio de alguna de las dos formas mencionadas con anterioridad. Para ilustrar estas dos formas de operaciones se presenta la siguiente figura.

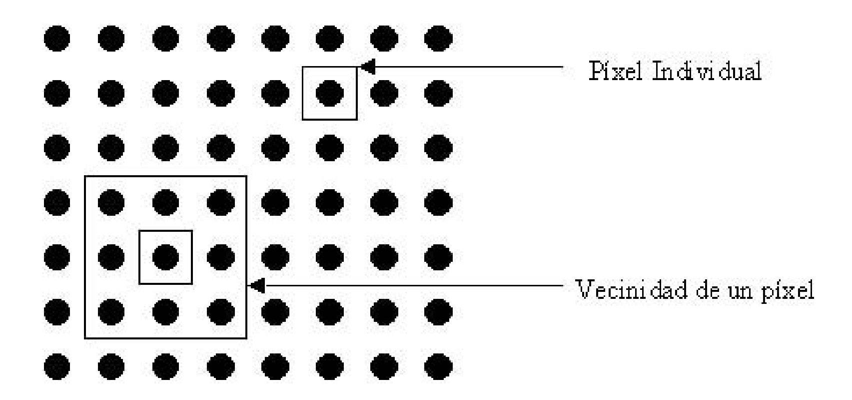


Figura 5.1 Operadores sobre píxeles.

5.2.1 Operador conversión.

Este operador es del tipo individual, es decir, de píxel a píxel. Su función es convertir una imagen de color en una matriz de intensidad de gris, debido a esto el método recibe como parámetro la imagen a convertir y regresa una matriz de dimensiones idénticas al tamaño de la imagen.

Este operador es la base para el funcionamiento del sistema debido a que todos los métodos implementados para el sistema llaman a éste método dado a que trabajan sobre matrices de intensidad y no directamente sobre las imágenes. Los valores de intensidad de dichas matrices pueden variar entre 0 y 255, donde el valor 0 corresponde al color negro y 255 al blanco.

```
try{
        PixelGrabber pg = new PixelGrabber(
                                        imagen.getSource(), 0, 0, ancho, alto, pix, 0, ancho);
        if(pg.grabPixels()){
                for(int i=0; i< alto*ancho; i++){
```

```
                    c[i] = new Color(pix[i]);
                    pixfin[i]= (int) (c[i].getRed()+c[i].getGreen()+c[i].getBlue())/3;
                    //pixfin[i]= (int) ( (c[i].getRed()*0.3) + (c[i].getGreen()*.59) +
                                        (c[i].getBlue()*.11));
                    c[i] = new Color(pixfin[i], pixfin[i], pixfin[i]);
                    pixfin[i] = c[i].getRGB();
            }
        }else{
            System.out.println("Problemas al descomponer la imagen");
        }
}catch(Exception e){
        System.out.println("Error "+e);
}
for(int i=0; i<alto; i++)
{
        for(int j=0; j<ancho; j++)
        {
            if(pixfin[cont]%256 == 0){
                    m[i][j] = 0;
            }else{
                    m[i][j] =  Math.abs(pixfin[cont]%256);
            }
            cont++;
        }
}
```

5.2.2 Operador de conversión de matriz a imagen.

Este operador recibe como parámetro una matriz con los valores de la escala de grises de cada uno de los píxeles y genera un objeto de tipo *Image* para poder ser desplegado por cualquier navegador de Internet.

```
Color c;
Image res;
for(int i=0; i<m.length; i++){
        for(int j=0; j<m[0].length;j++){
                c= new Color(m[i][j], m[i][j], m[i][j]);
                arreglo[cont] = c.getRGB();
                cont++;
        }
}
res = new Frame().createImage(new MemoryImageSource(ancho, alto, arreglo, 0, ancho));
```

5.2.3 Operador de Sobel.

Es un operador de extracción de bordes el cual toma en cuenta todos los píxeles vecinos del píxel de interés, por lo que se dice que es un operador de vecindad. Este operador funciona aplicando una máscara cuadrada de tamaño 3 X 3 sobre todos los píxeles de la imagen. Por medio de la mascara de Sobel se obtienen los valores de G_x y G_y mediante las siguientes formulas obtenidas en [Pajares, et al., 2004]:

$$Gx = (z_1 + 2z_4 + z_7) - (z_3 + 2z_6 + z_9)$$

$$G_y = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$$

Donde los valores de z corresponden a la intensidad de gris de los píxeles vecinos. Las máscaras para obtener G_x y G_y respectivamente son:

$$\begin{bmatrix} z1 & z2 & z3 \\ z4 & z5 & z6 \\ z7 & z8 & z9 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Una vez obtenidos los valores de G_x y G_y, se obtiene la magnitud del gradiente por medio de la función:

$$|G| = \sqrt{G_x^2 + G_y^2}$$

```
m = ti.convierte(imagen);
for(int r = 1; r < m.length - 1; r++){
        for(int c = 1; c < m[0].length - 1; c++){
                int xtemp = m[r-1][c-1] +
                        2*m[r][c-1] +
                        m[r+1][c-1] -
                        2*m[r-1][c-1] -
                        m[r][c] -
                        m[r+1][c+1];
                int ytemp = m[r-1][c-1] +
                        2*m[r-1][c] +
                        m[r-1][c+1] -
                        m[r+1][c-1] -
                        2*m[r+1][c] -
                        m[r+1][c+1];
                matFin[r][c] = (int) (Math.sqrt(xtemp*xtemp + ytemp*ytemp)%256);
        }
}
```

5.2.4 Operador de Prewitt.

El operador de Prewitt es muy similar al operador de Sobel, ambos trabajan con los píxeles vecinos del píxel de interés y aplicando una máscara cuadrada de 3 X 3. Lo que diferencia al operador de Prewitt del de Sobel, son los coeficientes de las máscaras para obtener los valores de G_x y G_y, cuyos valores se obtuvieron en [Pajares, et al., 2004] y se presentan a continuación:

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \qquad \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

La forma de calcular la magnitud del gradiente es de la misma manera que con el operador de Sobel:

$$|G| = \sqrt{G_x^2 + G_y^2}$$

```
m = ti.convierte(imagen);
for(int r = 1; r < m.length - 1; r++){
        for(int c = 1; c < m[0].length - 1; c++){
                int xtemp = m[r-1][c-1] +
                        m[r][c-1] +
                        m[r+1][c-1] -
                        m[r-1][c-1] -
                        m[r][c] -
                        m[r+1][c+1];

                int ytemp = m[r-1][c-1] +
                        m[r-1][c] +
                        m[r-1][c+1] -
```

```
                    m[r+1][c-1] -
                    m[r+1][c] -
                    m[r+1][c+1];

            matFin[r][c] = (int) (Math.sqrt(xtemp*xtemp + ytemp*ytemp)%256);

    }
}
```

5.2.5 Operador de Roberts.

Es un operador de vecindad al igual que los dos anteriores, sin embargo, Roberts marca solamente los puntos de borde, sin informarnos sobre la orientación de éstos. Es el operador de detección de bordes más simple el cual trabaja muy bien para imágenes binarias. Su funcionamiento se basa en el cálculo de las fórmulas siguientes, las cuales fueron obtenidas en [Pajares, et al., 2004]:

$$D_1 = z_4 - z_1 \text{ y } D_2 = z_2 - z_4$$

Una vez calculados los valores de las diagonales se puede obtener el valor resultante de dos formas, para la implementación del sistema se utilizó la primera de ellas.

$$R = \sqrt{D_1^2 + D_2^2} \text{ ó } R = |D1| + |D2|$$

```
m = ti.convierte(i);
for(int r = 1; r < m.length - 1; r++){
    for(int c = 1; c < m[0].length - 1; c++){
        int xtemp = m[r-1][c] -m[r][c-1];
        int ytemp = m[r][c] -m[r-1][c-1];
```

```
        matFin[r][c] = (int) (Math.sqrt(xtemp*xtemp + ytemp*ytemp)%256);
    }
}
```

5.2.6 Algoritmo SUSAN.

El algoritmo para detección de puntos esquina SUSAN es un operador de vecindad de píxeles puesto que arrastra una máscara circular de 37 píxeles por toda la imagen para determinar el brillo del píxel de interés. La forma de generar la máscara circular es:

```
for(int i=0; i<r; i++){
    for(int j=0; j<r; j++){
        mascara[i][j] = 1;
    }
}
mascara[0][0] = 0;
mascara[0][1] = 0;
mascara[0][5] = 0;
mascara[0][6] = 0;
mascara[1][0] = 0;
mascara[1][6] = 0;
mascara[5][0] = 0;
mascara[5][6] = 0;
mascara[6][0] = 0;
mascara[6][1] = 0;
mascara[6][5] = 0;
mascara[6][6] = 0;
```

Una vez generada la máscara se compara contra todos lo píxeles de la imagen para determinar el área USAN la cual muestra las diferencias entre los brillos de los puntos que caen dentro de la máscara con el píxel central de la máscara, el cual es el píxel de interés. El área USAN se calcula de la siguiente manera:

```
for(int i=1; i<m.length-1; i++){
    for(int j=1; j<m[0].length-1; j++){
        rengm = 0;
        colm=0;
        valor =0.0;

        for(int reng= i-(int)radio/2; reng <= i+(int)radio/2; reng++){
            colm=0;
            for(int col= j-(int)radio/2; col <= j+(int)radio/2; col++){
                if( (reng>=0) && (reng<m.length) && (col>=0) &&    (col<m[0].length) ){
                    pertenece = Math.abs( (m[reng][col]-m[i][j]) ) <= t;
                    if(pertenece){
                        if(masc.mascara[rengm][colm]==1){
                            if(Math.abs(m[reng][col]-m[i][j]) <= t){
                                double a = (Math.pow(((double)(m[reng][col])-(double)(m[i][j]))
                                / (double)t, (double)6d))*-1;
                                double exp = Math.exp(a);
                                valor = valor + exp;
                            }
                        }
                    }
                }
            colm++;
            }
    rengm++;
```

```
    }
    r[i][j] = (int)valor;
    }
}
```

Cuando se obtiene el área USAN para la imagen a tratar, ya se pueden determinar los puntos esquina verificando que estos últimos no sean puntos falsos, el siguiente fragmento de código muestra esta función.

```
for(int i=0; i<m.length; i++){
        for(int j=0; j<m[0].length; j++){

                if( (m[i][j]>0) && (m[i][j]<g) ){
                        r[i][j] = g-m[i][j];
                }
                else{
                        r[i][j] = 0;
                }

        }
}
for(int i=0; i<m.length; i++){
        for(int j=0; j<m[0].length; j++){

                if( m[i][j]> mayor ){
                        mayor = m[i][j];
                }
                if( m[i][j]<= menor ){
                        menor = m[i][j];
                }
        }
```

```
}
int ptos=0;
for(int i=0; i<m.length; i++){
	for(int j=0; j<m[0].length; j++){
		if (m[i][j] >= (mayor - dist)){
			ptos++;
			r[i][j]= 1;
		}
		else{
			r[i][j]= 0;
		}
	}
}
```

5.3 Interfaces del sistema.

Como se mencionó en el capítulo anterior en el apartado de casos de uso del sistema, éste tiene tres principales usos, para cada uno de los cuales la vista del sistema presenta una interfaz de usuario sencilla para que el usuario la llene según la petición que desee realizar.

Las primeras dos interfaces tienen una función similar, una es para cuando el usuario sólo desea subir una imagen al servidor para procesarla posteriormente, mientras que la segunda interfaz se presenta cuando el usuario desea subir al servidor más de una imagen. Para el caso de una imagen se presenta la siguiente interfaz:

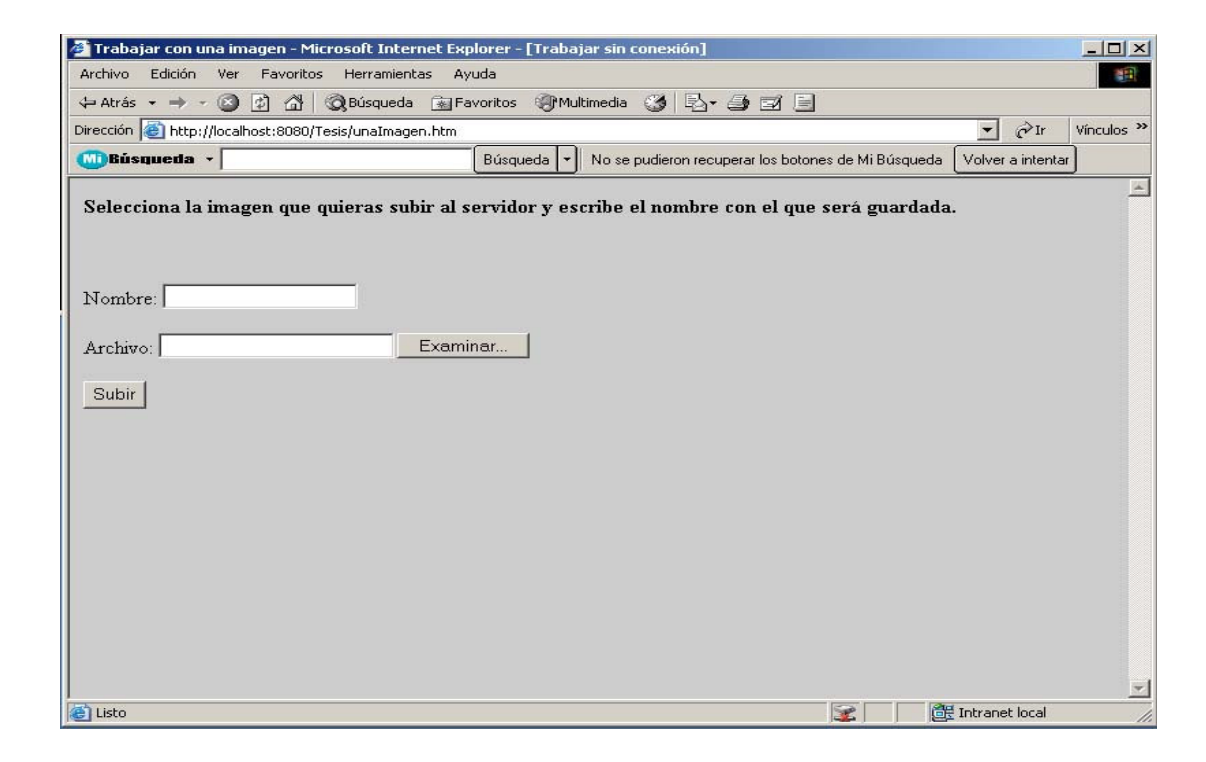


Figura 5.2 Interfaz para subir una imagen-

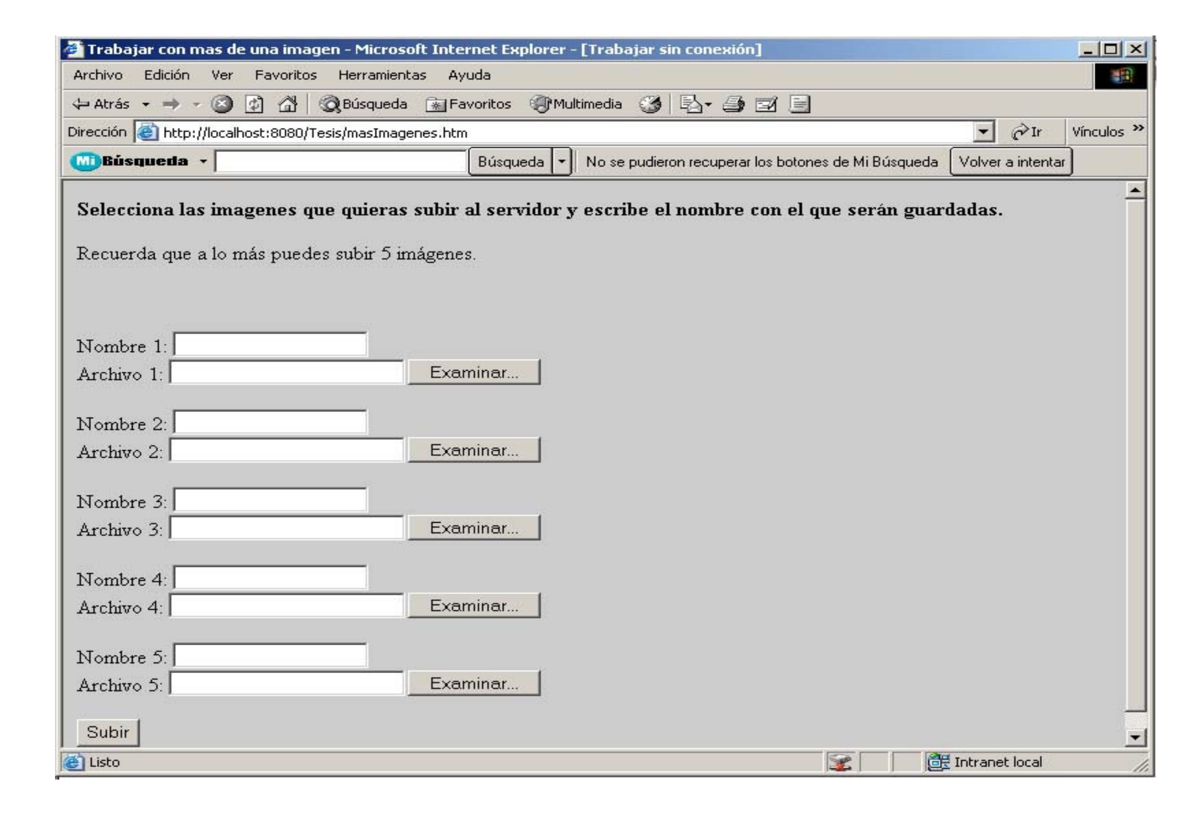


Figura 5.3 Interfaz para subir más de una imagen.

Otra interfaz que proporciona el sistema es cuando el usuario no desea subir una imagen al servidor, sino que quiere trabajar con una o más imágenes que han sido cargadas previamente. En este caso el sistema provee de la siguiente interfaz:

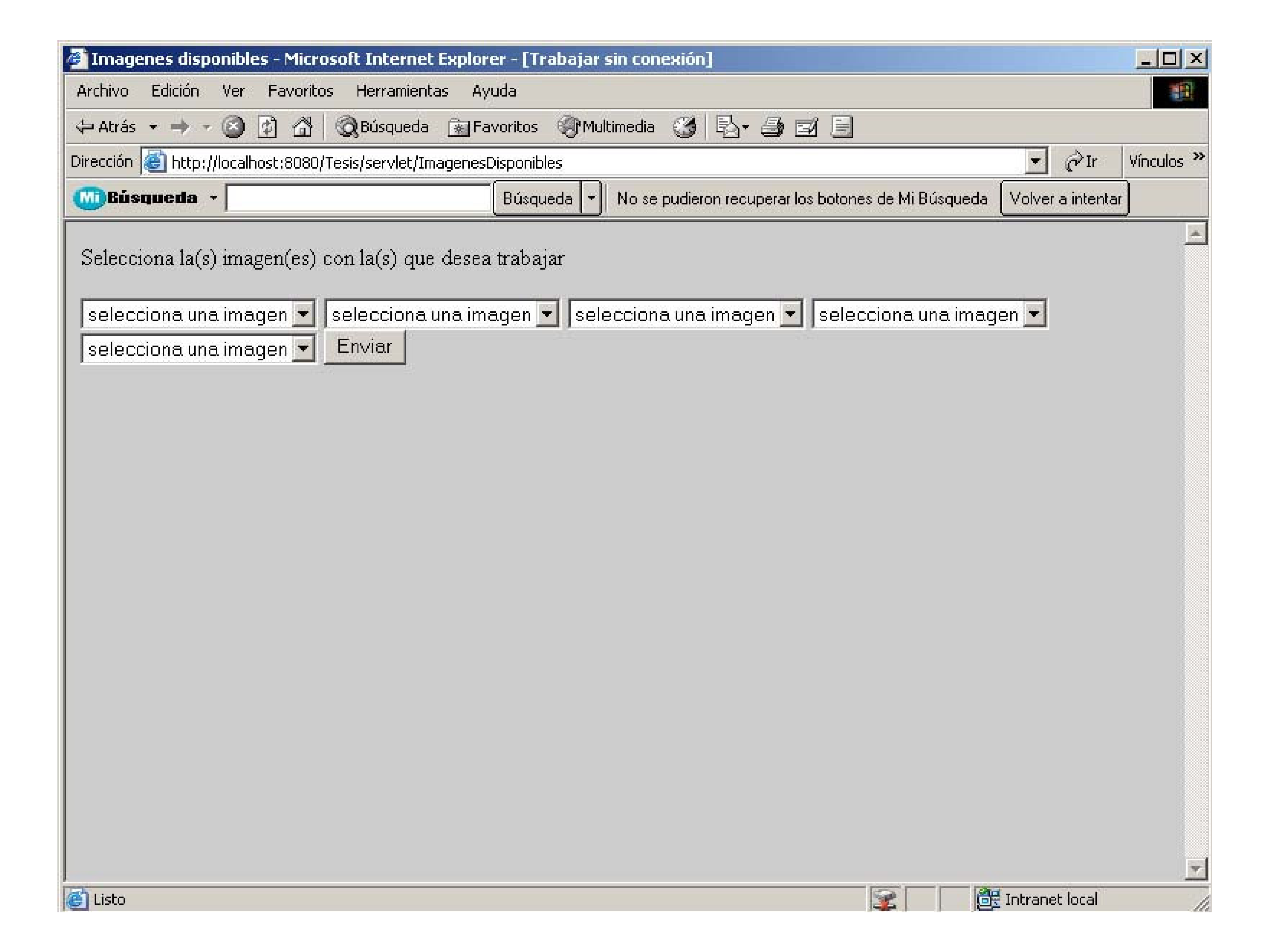


Figura 5.4 Interfaz para seleccionar imágenes disponibles.

Una vez que el usuario cargó la imagen que desea procesar, se presenta otra interfaz en la cual el usuario deberá seleccionar tanto el operador que quiere aplicarle a la imagen seleccionada como los parámetros necesarios para dicho operador. Esta interfaz se presenta a continuación:

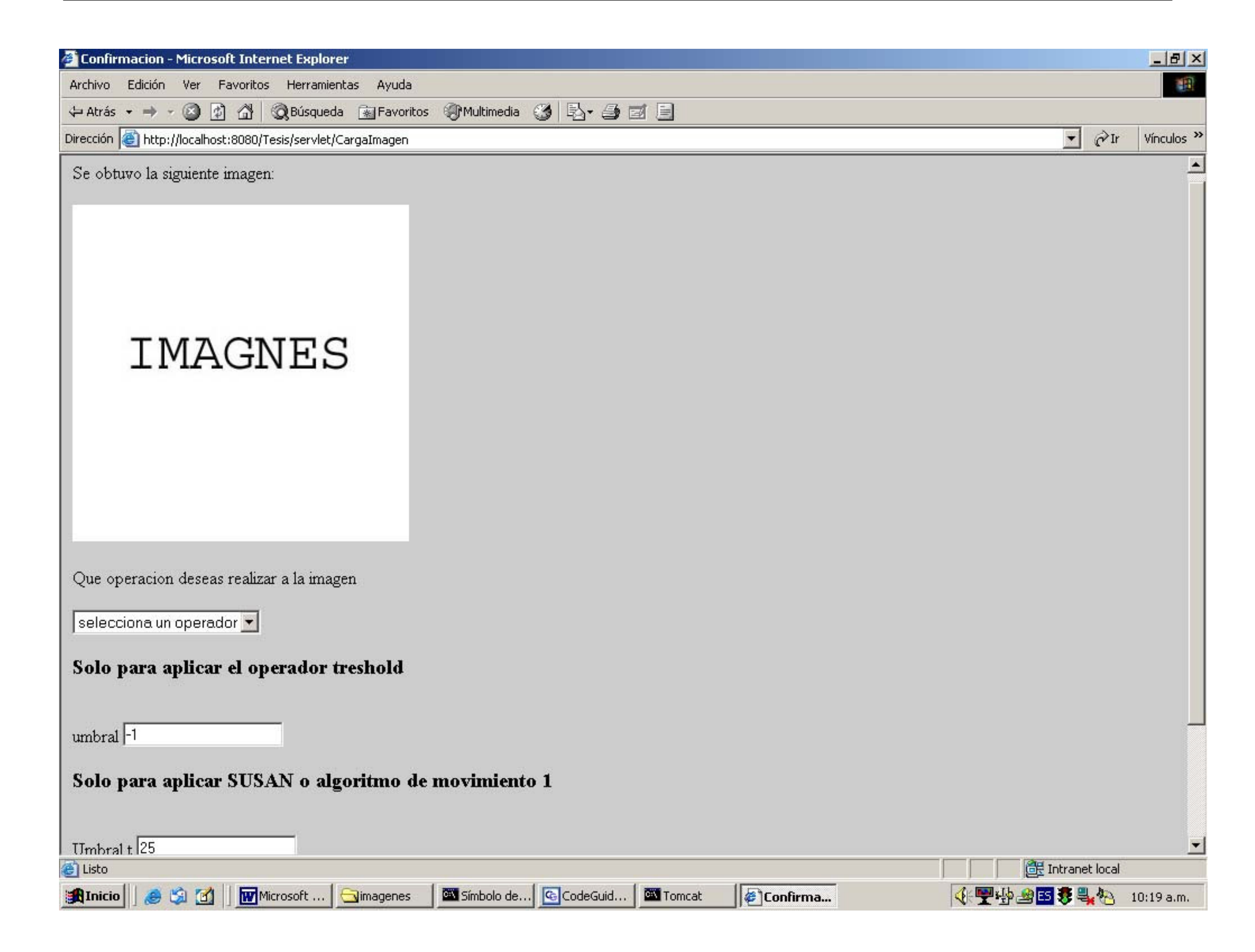


Figura 5.5 Interfaz para seleccionar operador disponible.

5.4 Conclusiones.

Este capítulo muestra más a fondo los tres grupos que componen el sistema, muestra tanto las interfaces de usuario como los módulos de código más importantes de los operadores que forman el sistema.

Este capítulo no explica por completo el funcionamiento del sistema, sin embargo, da a conocer la forma en que trabaja cada operador sobre una imagen. Además, se presentan las interfaces de usuario que el sistema proporciona al usuario para cada una de

las posibles solicitudes de dicho usuario. Para mayor explicación sobre el correcto uso del sistema, revísese el Apéndice A correspondiente al manual de usuario, para ver la implementación de los demás operadores, revísese el Apéndice C correspondiente a operadores adicionales y para ver algunos resultados obtenidos por todos los métodos que provee el sistema revísese el Apéndice D correspondiente a pruebas adicionales.